**A PROPOSED OFFERING OF DENTAL CLINIC**

**INVENTORY AND SUPPLY MANAGEMENT SYSTEM FOR**

**MCVIL**

Prepared by: Angelo, Bugawan A.

BSIT 3rd year C

September, 2025

**Design Document**

**INTRODUCTION**

**Purpose of the Document**

The purpose of this Software Design Document (SDD) is to provide a comprehensive blueprint for the development, deployment, and maintenance of the Dental Clinic Inventory and Supply Management System (DCISMS). This document ensures that all stakeholders, including developers, testers, and clinic staff, have a clear understanding of the system’s structure, features, and implementation strategies. It acts as a bridge between the requirements specification and the actual coding process, reducing ambiguity and miscommunication during development.

**Overview of the Software System**

The DCISMS is a standalone software application designed for Macvil Dental Clinic. Its primary function is to streamline the tracking and management of dental clinic supplies and inventory. It replaces manual record-keeping with a digitized solution that allows for real-time monitoring of stock levels, item categorization, transaction logging, and secure user access. The system supports administrative staff by reducing human errors, minimizing supply shortages, and increasing operational efficiency.

**Scope of the Document**

This document provides detailed coverage of the system architecture, database schema, user interface design, and component-level specifications. It also addresses security measures, performance considerations, error handling, and deployment guidelines. By covering all these aspects, the document ensures that future developers and maintainers can extend and update the system without compromising its integrity.

**SYSTEM ARCHICTURE**

The DCISMS follows a modular client-server architecture where the frontend interfaces with backend logic through HTTP requests. The backend is powered by Flask (Python), and data is persisted in an SQLite database.

**High-Level Components and Interactions**

1. **Frontend Layer** – Developed using HTML, CSS, and Bootstrap for responsive design. Handles user input, displays data, and communicates with the backend through Flask routes.
2. **Backend Layer** – Flask-based server application that handles business logic, data validation, authentication, and authorization.
3. **Database Layer** – SQLite database that stores persistent data such as user credentials, inventory items, categories, and transaction history.

**Deployment Architecture**

* **Standalone Mode**: Runs locally on a desktop or laptop with Python 3.13 and SQLite installed.
* **Server Deployment**: Can be deployed on a Linux/Windows server for multi-user access within a clinic’s intranet.

**Communication Protocols and Interfaces**

* Frontend to Backend: HTTP/HTTPS requests handled by Flask routes.
* Backend to Database: SQL queries via SQLite driver.

**DATABASE DESIGN**

Entity-Relationship Diagram (ERD)  
The database consists of four main entities: Users, Items, Categories, and Transactions. Relationships are established to maintain referential integrity.

**Tables and Fields**

* Users Table (users)
  + user\_id (Primary Key)
  + username
  + password\_hash
  + role (admin, staff)
* Items Table (items)
  + item\_id (Primary Key)
  + name
  + category\_id (Foreign Key → categories)
  + quantity
  + description
* Categories Table (categories)
  + category\_id (Primary Key)
  + category\_name
* Transactions Table (transactions)
  + transaction\_id (Primary Key)
  + item\_id (Foreign Key → items)
  + user\_id (Foreign Key → users)
  + action (added, updated, removed)
  + quantity
  + timestamp

**Normalization Techniques**

* First Normal Form (1NF): No repeating groups, atomic fields.
* Second Normal Form (2NF): Non-key fields depend fully on primary keys.
* Third Normal Form (3NF): No transitive dependencies.

**USER INTERFACE**

**Wireframes and Mockups**

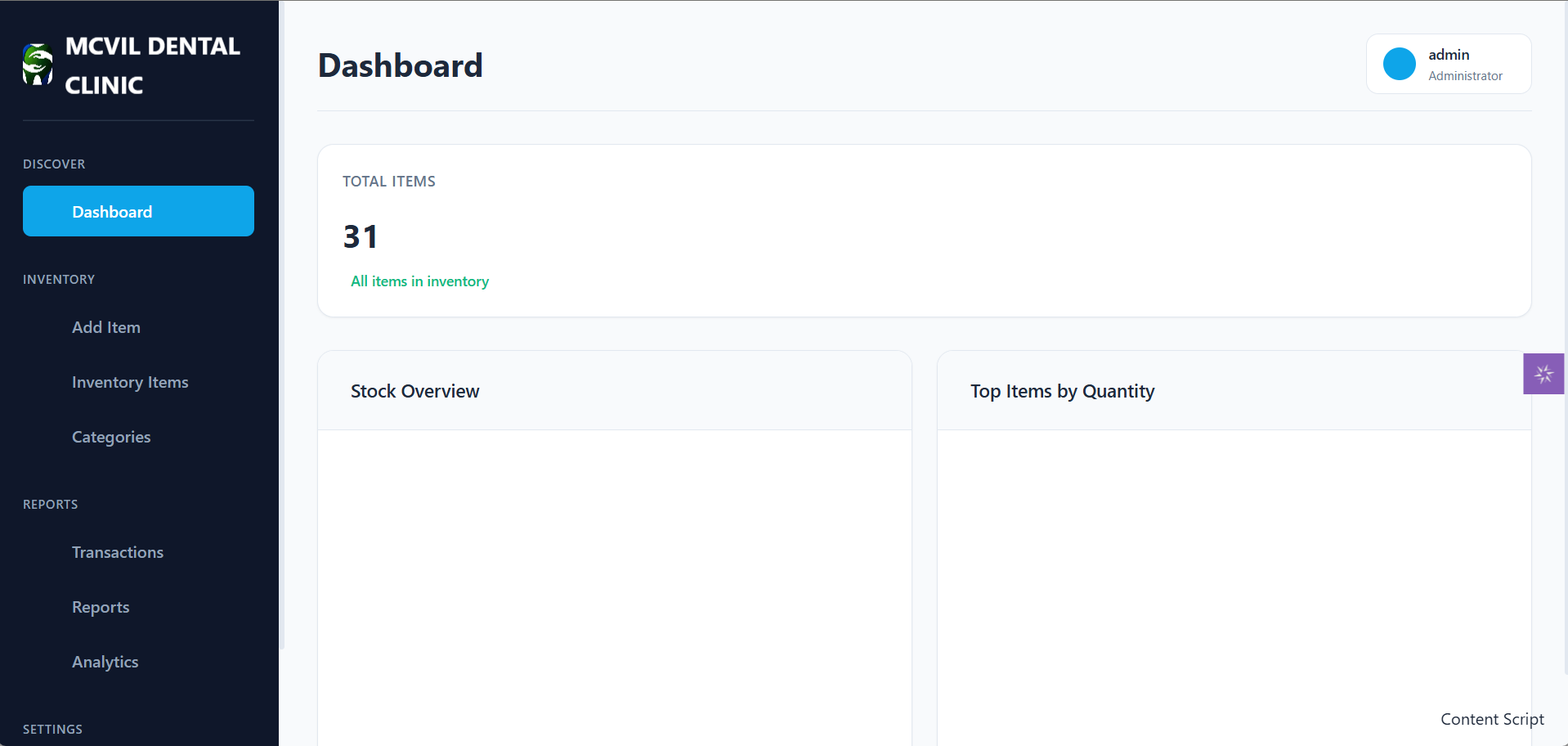
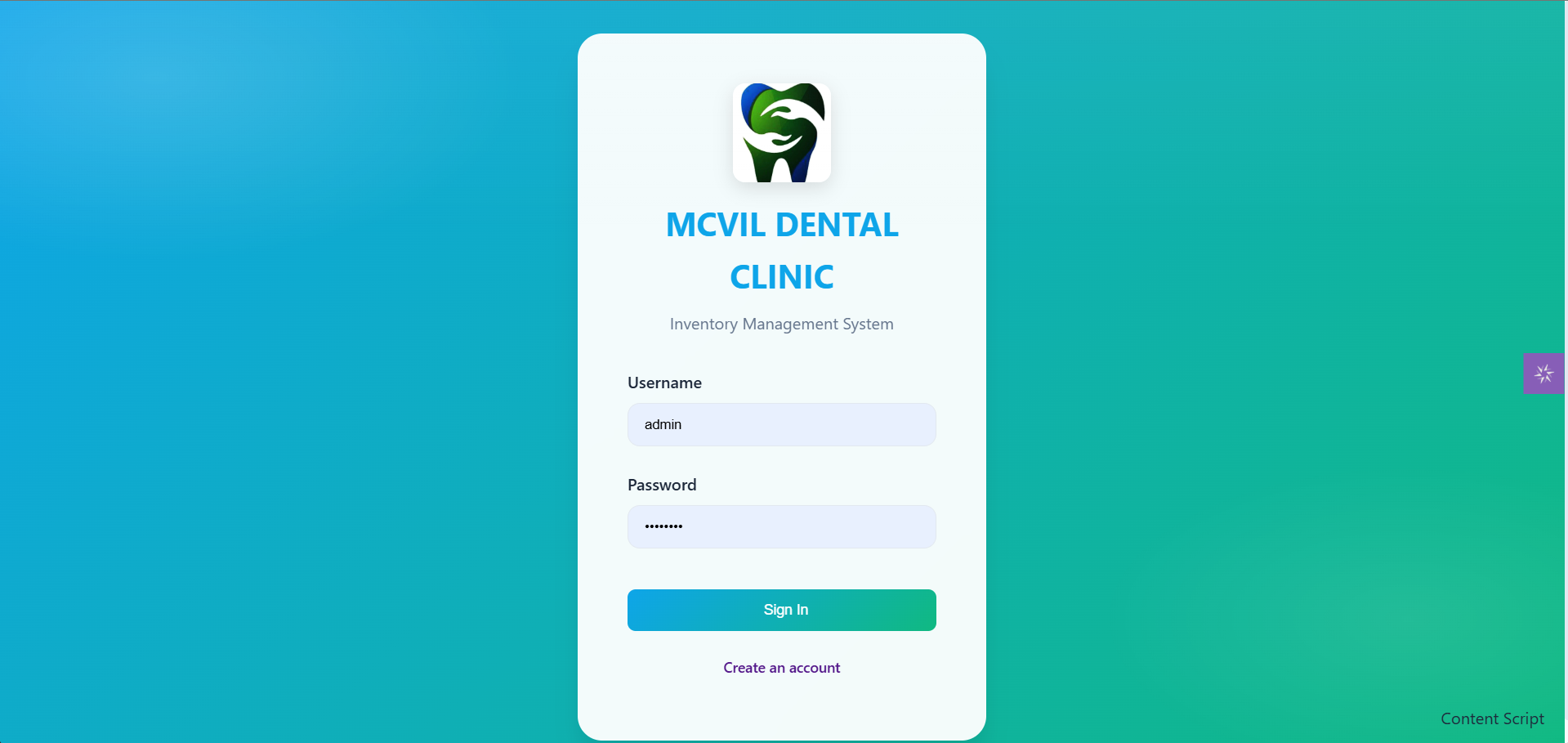
* Login Page: Clean, centered login form with username and password fields.
* Dashboard: Displays stock summaries, low-stock alerts, and recent transactions.
* Add Item Page: Input form for adding new supplies with category dropdown.
* Edit Item Page: Allows modification of item details.

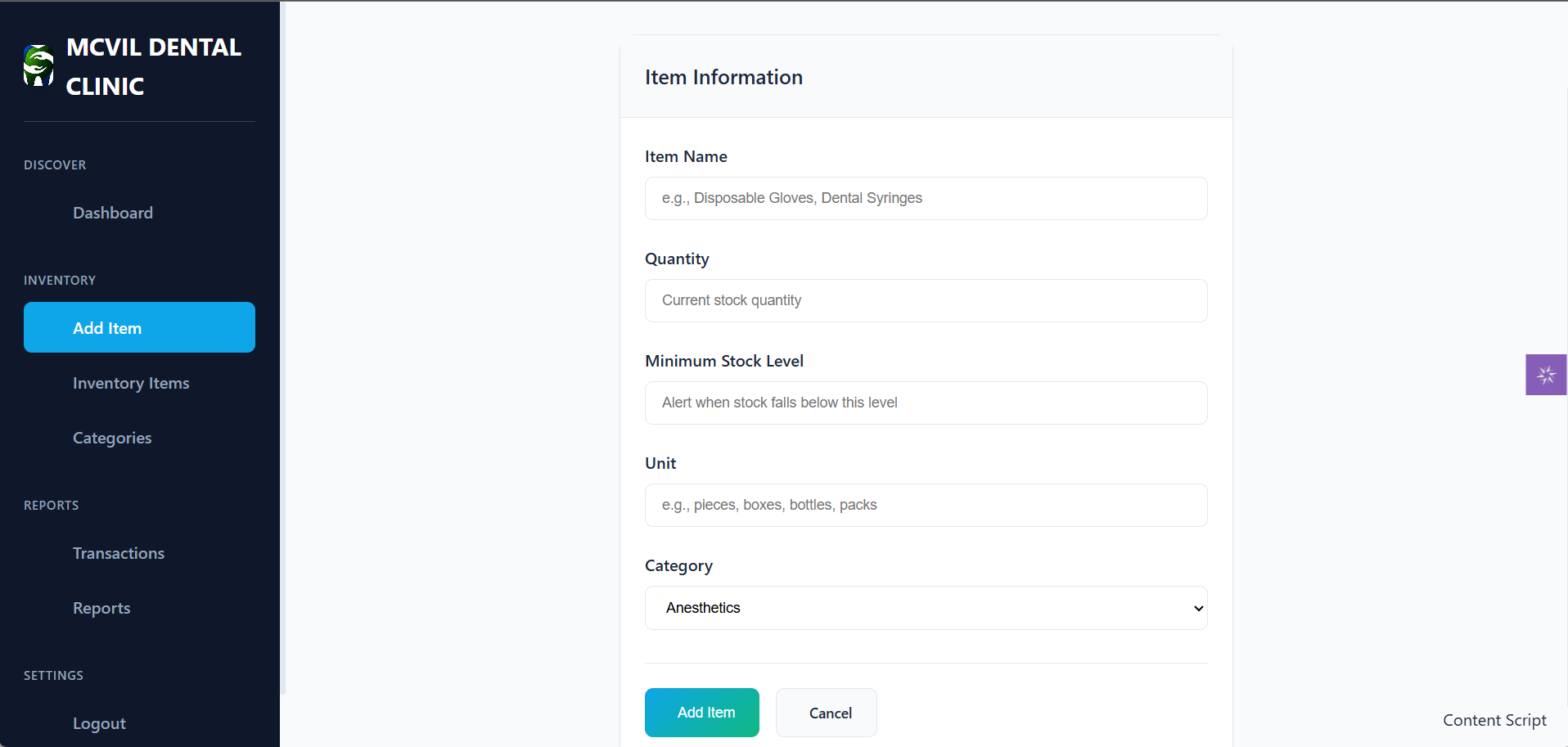
**UI Elements and Layout**

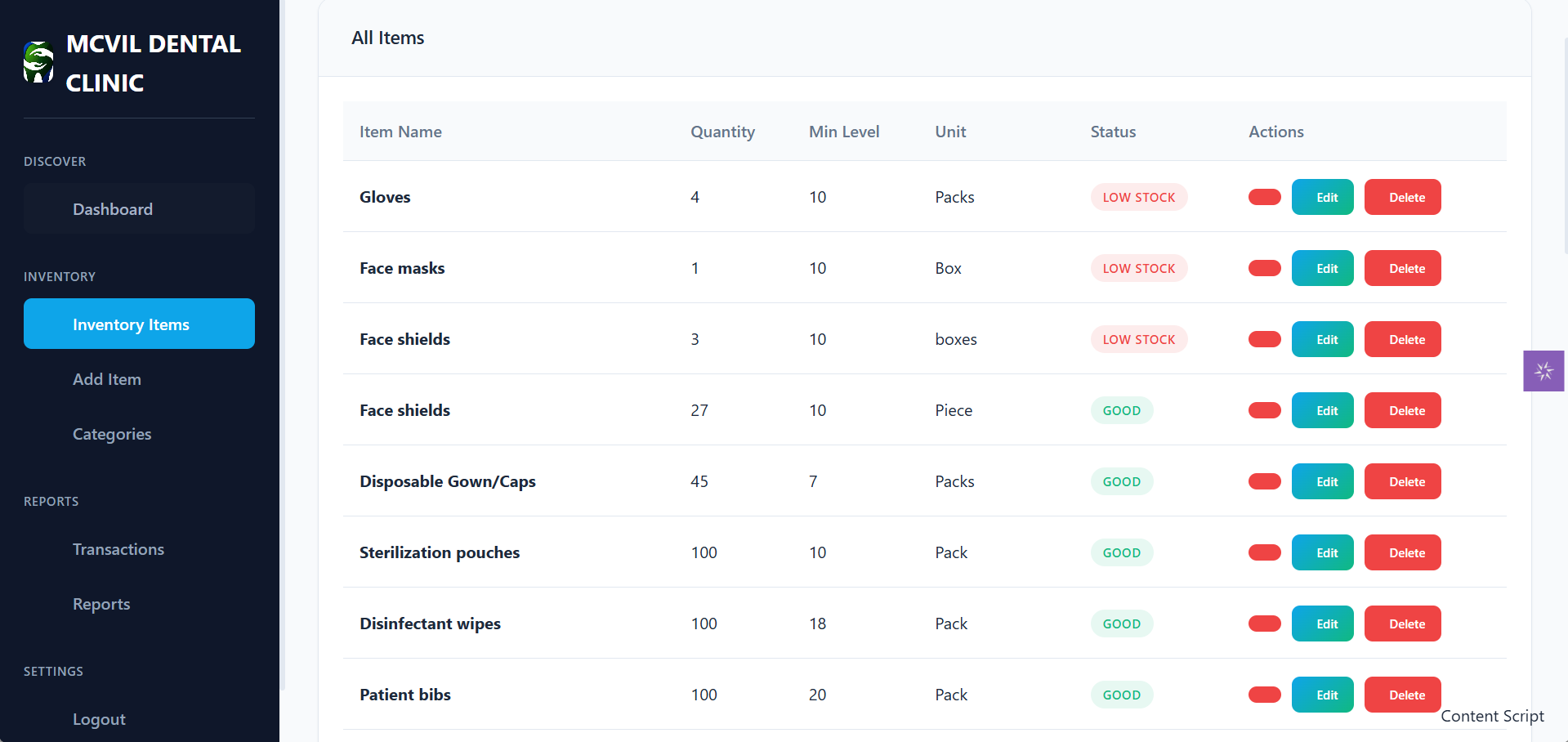
* Navigation Bar: Provides quick access to inventory, categories, transactions, and logout.
* Forms: Used for login, item entry, and editing.
* Tables: Display items and transactions in structured lists.

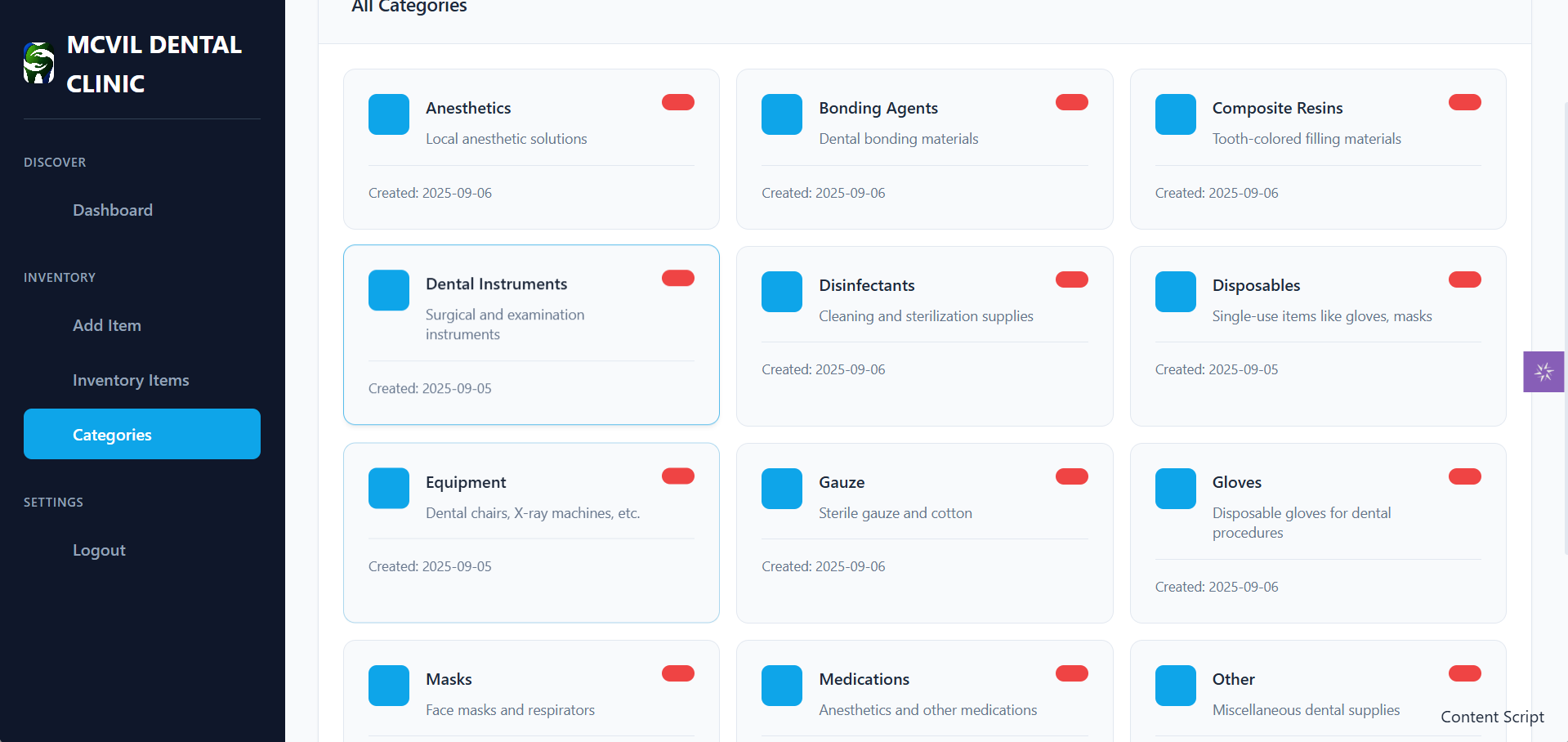
**Usability Considerations**

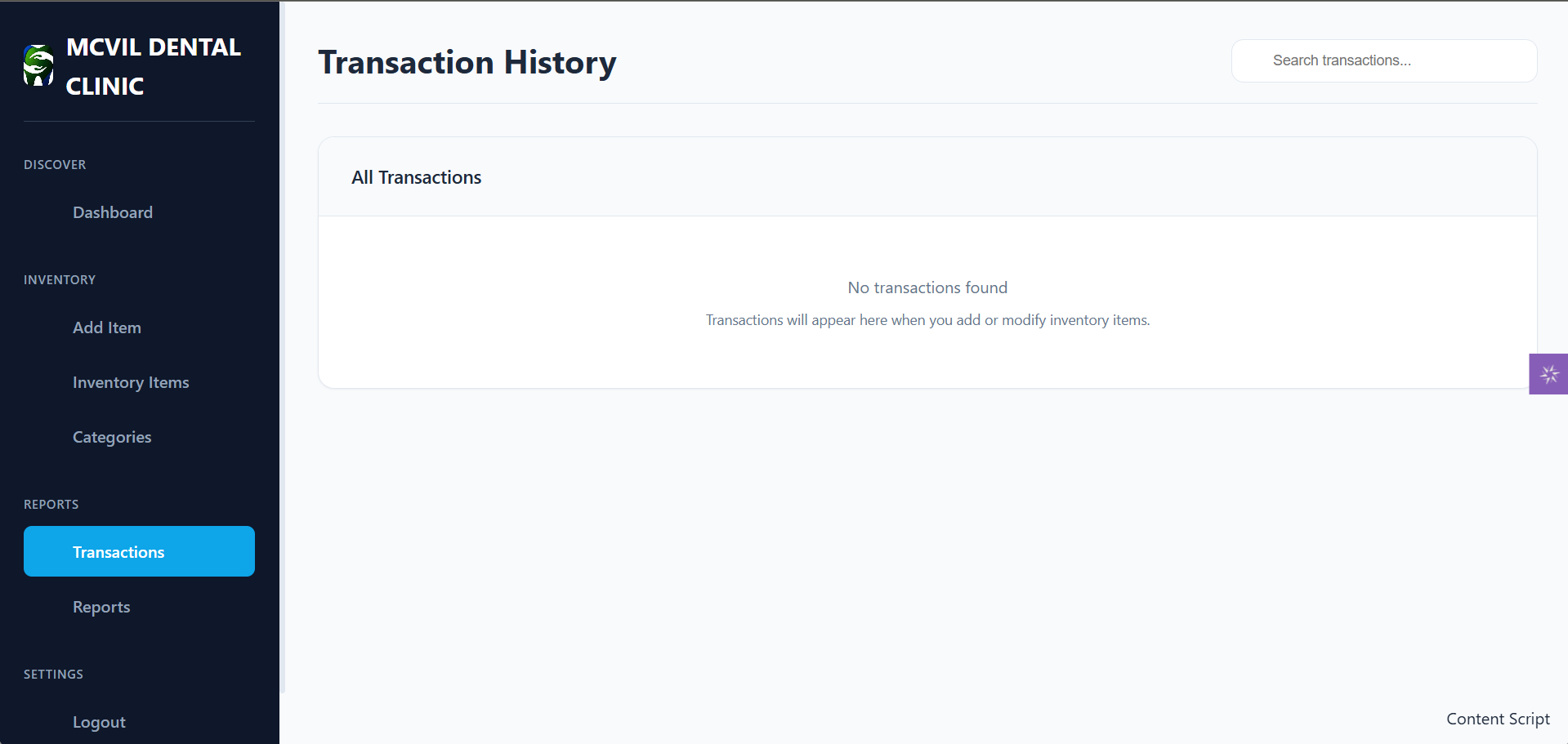
* Mobile responsiveness for tablets and smaller screens.
* Accessibility features including ARIA labels and keyboard navigation.
* Consistent color scheme aligned with the clinic’s branding.



****

****







**COMPONENT DESIGN**

**Key Components/Modules**

1. Authentication Module – Handles login, logout, and role-based access control.
2. Inventory Module – CRUD operations for managing items.
3. Category Module – Defines and manages categories for classification.
4. Transaction Module – Logs and displays actions taken on inventory.
5. Dashboard Module – Provides an overview of system statistics.

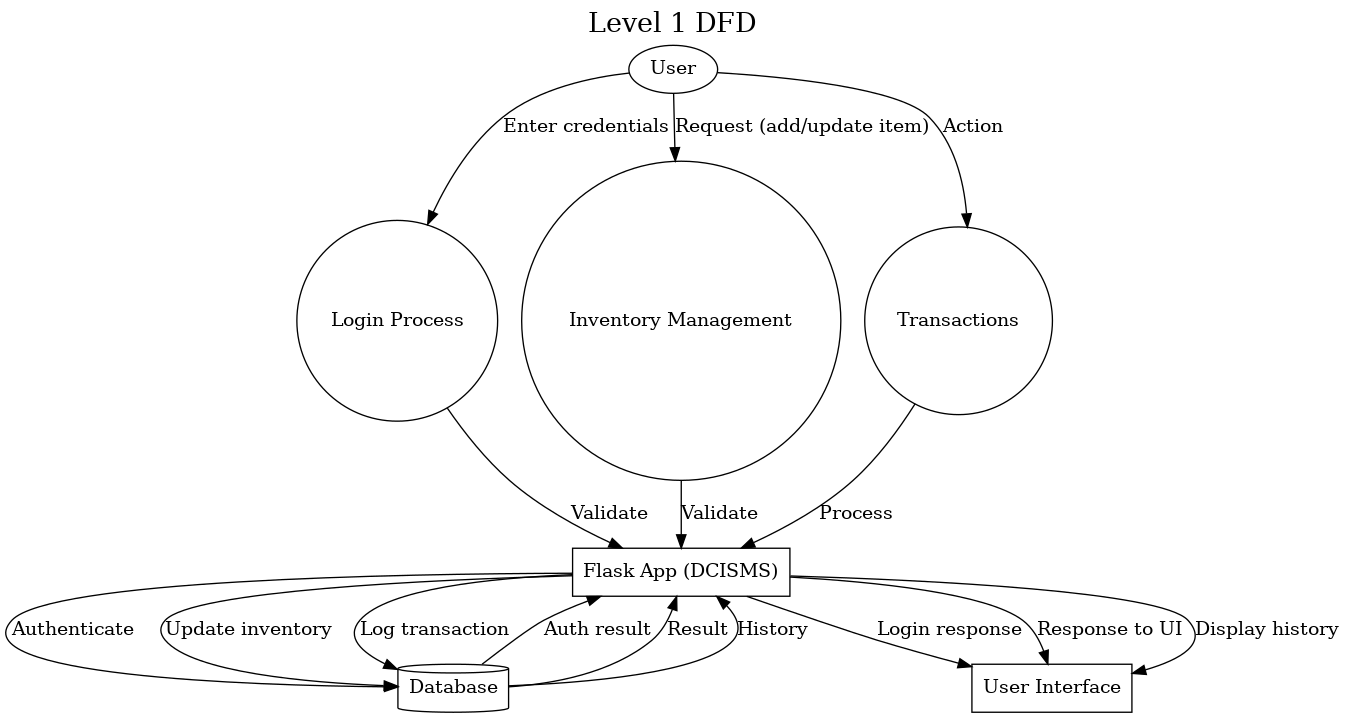
**Interface Specifications**

* Flask Routes: /login, /dashboard, /add\_item, /edit\_item/<id>, /delete\_item/<id>, /transactions.
* Functions: add\_item(), edit\_item(), delete\_item(), get\_transactions().

**Dependencies**

* Flask (web framework)
* SQLite3 (database engine)

**DATA FLOW DIAGRAM**

****

**SECURITY DESIGN**

**Requirements**

* Secure authentication
* Data confidentiality and integrity
* Role-based authorization

**Mechanisms**

* Session-based authentication tokens.

**Data Encryption and Protection**

* HTTPS protocol for secure data transmission.
* Database access restricted by role.

**PERFORMANCE DESIGN**

**Performance Requirements**

* CRUD operations must complete under 1 second.
* System must handle at least 100 concurrent transactions without performance degradation.

**Optimization Strategies**

* Database indexing for frequent queries.
* Query optimization for SELECT, UPDATE operations.
* Option for in-memory caching if scaled to cloud deployment.

**Performance Testing Plan**

* Unit tests for query performance.
* Load testing with simulated multiple users.
* Stress testing for peak usage scenarios.

**ERROR HANDLING AND LOGGING**

**Error Handling Mechanisms**

* Flask try-except blocks for handling runtime errors.
* User-friendly error pages (404, 500).

**Logging Specifications**

* Log user actions with timestamps.
* Log errors to a file with severity levels (INFO, WARNING, ERROR).

**Error Codes**

* 400: Bad request
* 401: Unauthorized access
* 404: Page not found
* 500: Internal server error

**THIRD PARTY INTEGRATION**

* Flask: Web framework for backend logic.
* SQLite: Lightweight database for data persistence.
* Bootstrap: For frontend styling and responsiveness.
* Jinja2: Template engine for rendering dynamic HTML**.**

**DEPLOYMENT PLAN**

**Process**

1. Development in local environment.
2. Version control via GitHub.
3. Deployment on clinic server or workstation.
4. Optional: Cloud deployment for multi-branch access.

**Hardware Requirements**

* Minimum: Dual-core CPU, 4GB RAM, 500MB disk space.
* Recommended: Quad-core CPU, 8GB RAM.

**Software Requirements**

* Python 3.13
* Flask, SQLite, Bootstrap
* Web browser (Chrome, Firefox, Edge)

**MAINTENANCE AND SUPPORT**

**System Maintenance Guidelines**

* Regular database backups.
* Security patches applied monthly.

**Update Procedures**

* Re-run database migrations if applicable**.**

**Escalation Process**

* Tier 1: Clinic admin troubleshooting.
* Tier 2: Developer intervention.